Stochastic Gradient Descent

STAT 672 Project

Tom Wallace

George Mason University

Spring 2018

Estimating model coefficients requires optimization

We typically fit statistical models to maximize some measure of goodness (e.g., likelihood) or minimize some measure of badness (e.g., risk)

Parametric assumptions can make this optimization "nice"

OLS has a closed form solution: $\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})\mathbf{X}'\mathbf{Y}$

GLM often uses Newton's method

- Knowing the PDF makes finding the Hessian (2nd order derivatives) not too onerous

Non-parametric & high-dimensional settings

Suppose that we have a typical supervised classification problem

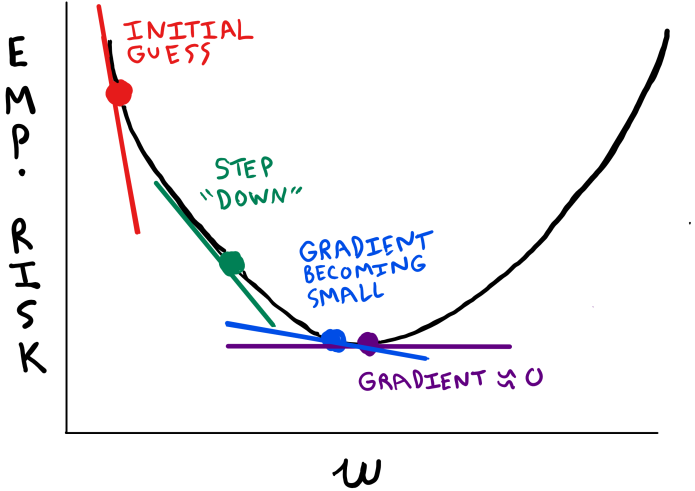
- Non-parametric: no assumptions about distribution of data
- Feature vector $\mathbf{X}_i$, label Y_i
- Want to find best prediction function $f^*(\mathbf{w}; \mathbf{X})$ from hypothesis class $\mathcal{F}$
- Optimization: pick weights $\hat{\mathbf{w}}$ that minimize empirical risk according to some convex loss function

Some familiar tools for finding $\hat{\mathbf{w}}$ may no longer work well

But we still know that if L is convex, there is a unique global minimum, and the gradient at that point must be 0

Gradient descent is a numerical strategy to search for that point

Gradient descent visualized



A more formal explanation

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_w L(f(\mathbf{w}^{(t)}; \mathbf{X}_i), Y_i)$$

Stop if $||\text{gradient}||_2 \leq \epsilon$

Step size γ can vary over time

Batch gradient descent is computationally expensive

In “plain” (batch) gradient descent, for every step, we have to evaluate the gradient at every observation

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_w L(f(\mathbf{w}^{(t)}; \mathbf{X}_i), Y_i)$$

This becomes computationally intractable as n grows large

- If $n = 10$ million, have to evaluate gradient 10 million times every step

Stochastic gradient descent (SGD) takes less time

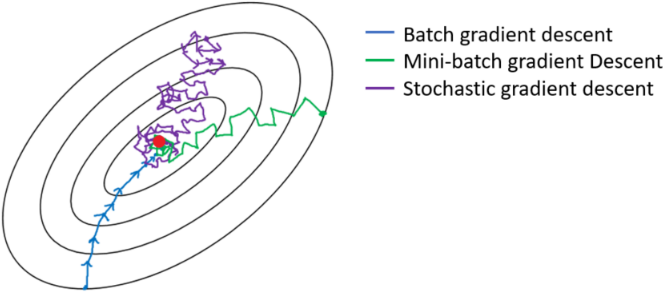
For each step, gradient is computed for a single randomly chosen observation i:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \gamma \nabla_w L(f(\mathbf{w}^{(t)}; \mathbf{X}_i), Y_{i)}$$

This simplification makes approximation much "noisier", and hence SGD requires more iterations

But, each iteration is faster and so SGD can reach a predefined level of risk or error in less time

SGD is noisier than batch GD

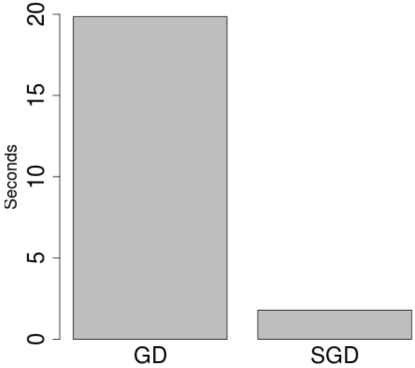


Source: towardsdatascience.com

SGD is useful when n is large & compute time is important

	GD	**SGD**
Time to accuracy ρ	$n \log \frac{1}{\rho}$	$\frac{1}{\rho}$

Figure: Fitting OLS coefficients, $n = 10$ million



SGD is very popular in academia and industry

Many statistical learning models use SGD to fit weights

Particularly popular for neural networks (backpropagation)

If a Silicon Valley press release uses any of the following phrases...

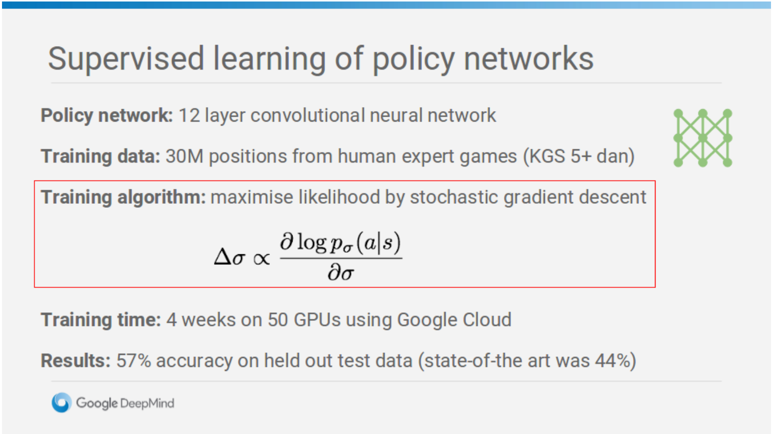
- "Machine learning"
- "AI"
- "Cognitive computing"

...SGD probably is involved.

Example: Google's **AlphaGo** program

nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
At last — a computer program that can beat a champion Go player PAGE 484
ALL SYSTEMS GO

Google slides from ICML 2016



Supervised learning of policy networks

Policy network: 12 layer convolutional neural network

Training data: 30M positions from human expert games (KGS 5+ dan)

Training algorithm: maximise likelihood by stochastic gradient descent

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial\sigma}$$

Training time: 4 weeks on 50 GPUs using Google Cloud

Results: 57% accuracy on held out test data (state-of-the art was 44%)

Google DeepMind